DOCUMENT RESUME

ED 427 692 IR 019 243

AUTHOR	Clyde, Stephen W.; Hirschi, Gregory W.
TITLE	Interactive Display of High-Resolution Images on the World Wide Web.
PUB DATE	1998-11-00
NOTE	7p.; In: WebNet 98 World Conference of the WWW, Internet, and Intranet Proceedings (3rd, Orlando, FL, November 7-12, 1998); see IR 019 231.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	Computer Interfaces; Computer Oriented Programs; Computer Software Development; *Computer System Design; Evaluation Criteria; Higher Education; Information Management; Information Technology; Interaction; User Needs (Information); *Visual Aids; *World Wide Web
IDENTIFIERS	*Client Server Computing Systems; *Digital Imagery; Interactive Systems; Utah State University

ABSTRACT

Viewing high-resolution images on the World Wide Web at a level of detail necessary for collaborative research is still a problem today, given the Internet's current bandwidth limitations and its ever increasing network traffic. ImageEyes is an interactive display tool being developed at Utah State University that addresses this problem by integrating caching, compression, and transmission techniques and performing global optimization across these techniques. It is an interactive image display facility that allows Web users to explore high-resolution images in near real-time without upgrading their computers or networks. Although much of ImageEyes is based on existing technology, it employs several innovations, including a three-dimensional, hierarchical cache and a data striping technique that tolerates lossy transmission without significant degradation to the user's view. This paper describes: the ImageEyes architecture, including Web components and protocols, client software, user interface, client's cache, and server's image processor; an operational overview; performance evaluation; and related work. A figure presents a high-level view of ImageEyes' architecture. (Author/AEF)

Interactive Display of High-resolution Images on the World-Wide Web

Stephen W. Clyde
Utah State University, Computer Science Department
Logan, UT 84322-4205 USA
Email: swc@stevec.cs.usu.edu

Gregory W. Hirschi
Utah State University, Computer Science Department
Logan, UT 84322-4205 USA
Email: ghirschi@acm.org

Abstract: Viewing high-resolution images on the World-Wide Web at a level of detail necessary for collaborative research is a still a problem today, given the Internet's current bandwidth limitations and its ever increasing network traffic. ImageEyes is an interactive display tool being developed at Utah State University that addresses this problem by integrating caching, compression, and transmission techniques and performing global optimization across these techniques. Although much of ImageEyes is based on existing technology, it employs several innovations, including a three-dimensional, hierarchical cache and a data striping technique that tolerates lossy transmission without significant degradation to the user's view.

1 Introduction

The advent and popularity of the World-Wide Web (Web) has created an effective medium for scientists to collaborate among themselves and disseminate information to the public. Applications that require ad-hoc inspection of high-resolution images, however, are still not feasible because of current bandwidth limitations and increasing network traffic. One such application involves on-line examination of x-ray images. To be useful, doctors must be able to examine any part of an x-ray image in as much detail as the original film would allow. Ideally, when accessing an x-ray, a doctor should first see a high-level view of the whole image and then, in near real-time, be able to zoom and pan through the image, examining minute details like hair-line fractures. Other applications include on-line study of biological specimens, historical documents, artwork, and satellite photographs.

Most of the images on the Web today range in size from one to several hundred kilobytes. Although such images can fill a standard computer screen, they do not provide enough information for inspection of fine details, like bone fractures. Simply using higher resolution images, which can be several hundred megabytes in size, does not solve the problem. Current Web protocols are based on full-image transfers, irrespective of transmission speed, the resolution of the user's screen, and the availability of local storage [HTTP 1998]. Not only can full transfers require an unacceptable amount of time, but users may not have enough local storage space to hold the image data. In addition, why should the local system even try to retrieve and store that much data when only a small fraction of it can be displayed at any given time.

One possible approach is to cut a high-resolution image into smaller views that focus on predetermined areas of interest within the picture. For example, a high-resolution image of a plant might be broken down into separate views that focus on leaf venation, leaf arrangement, and its reproductive parts. Although this approach keeps the data requirements and transfer times within reasonable limits, it severely constrains the user's ability to freely explore the original image. For collaborative research, it is critical that users be able to explore areas of an image that may not have been previously considered interesting. Furthermore, extracting predefined views from high-resolution images takes significant effort and requires those who prepare the views to have an in-depth knowledge of the subject area.

Many of the restrictions and decision-making problems of the prior approach can be eliminated by fully and automatically partitioning the original image into sub-images at some zoom factor and then repeating

this process for each sub-image until all available detail is fully exposed. This would result in a complete hierarchy of views. Each level in the hierarchy would represent the full image at a specific zoom factor. Zoom operations would be implemented as hyper-links between views on different levels, whereas pan operations would be implemented as hyper-links between views on the same level. Although this approach allows exploration of the entire image, it has several serious drawbacks. One problem is a potential for excessive data transfer, even when the user's Web browser has a local cache. This problem stems from the way browsers treat image files as autonomous entities. They cannot reuse bits and pieces of previous images. For example, if a user is looking at the edge of a leaf and then zooms out to see the whole leaf, the browser requests the transfer of a whole new image, even though a portion of that image was just on the screen. Another problem is that extensive zooming and panning of an image would eventually fill the browser's cache and thereby render the cache useless as soon as the user leaves the image. Finally, since the zoom and pan operations are implemented with hyper-links, the backtracking feature (the *Back* button) available in most browsers would cause the operations to be played in reverse, which can initiate additional unnecessary data transfers.

ImageEyes is an interactive image display facility that will allow Web users to explore high-resolution images in near real-time without upgrading their computers or networks. This paper describes an overview of its design and operation and the status of work in progress.

2 ImageEyes' Architecture

Figure 1 shows a high-level view of the ImageEyes' architecture. The boxes labeled *Web Server*, *Web Browser*, and *Fast CGI Interface* and the interconnecting lines represent existing Web components and protocols [Connelly 1998, HTTP 1998, NCSA 1998, W3C 1998]. ImageEyes will use this existing technology for displaying entry-points and initiating image viewing sessions. The remaining boxes and ellipses represent the ImageEyes server and client. At the core of their design is an integrated strategy for global optimization, across its image processing, compression, communication, caching, and display techniques. The basic idea is to use observed conditions in one area to tune performance in another. For example, noisy transmission might cause ImageEyes to alter how it packages and compresses individual blocks of data prior to transmission.

The ImageEyes client software is responsible for interacting with the user, sending image requests to the server, coordinating responses, managing a local cache, and displaying the current view. We have

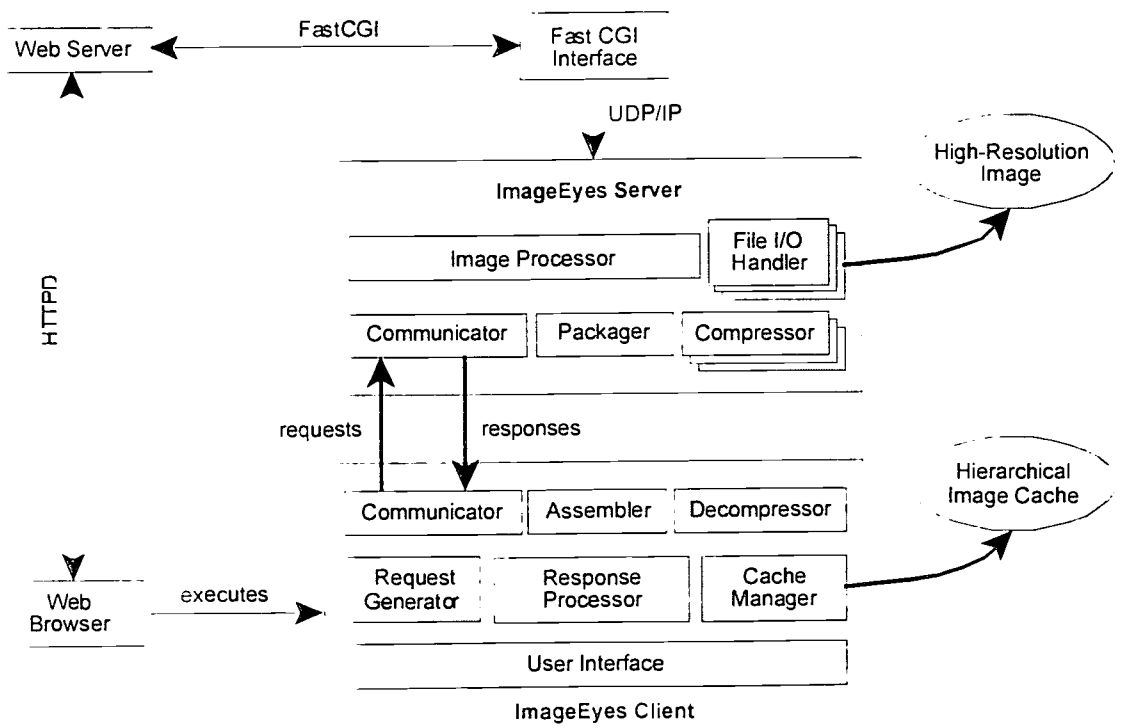

Figure 1: High -level view of ImageEyes' Architecture

implemented version 1.0 of the client in C++ as a stand-alone application running on HP-UX® with Motif 1.2®. Future implementations include plug-ins for Netscape® and Internet Explorer® and a Java applet. Because some of these variations are intended to be downloaded at run-time, our approach is to keep the client software as lightweight as possible. For this reason, all decision-making and optimization logic resides in the server.

The client's *user interface*, displays a view of the image and allows the user to navigate through the image with various zoom and pan operations. The *request generator* optimizes the navigational requests and send them to the server via a *communicator*. Each request includes information about the contents of the client's *cache* and current network conditions. The server uses this information to decide the minimum amount of data that has to be sent back and to optimize the packaging and compression of that data.

The client's *cache* is a hierarchical scheme consisting of logical layers, each representing the entire image for a different zoom factor. At any given time, a layer may contain no data, disjoint blocks of data, or a complete image. The top layer, which corresponds to a *1x*-zoom factor, is always completely in memory. Lower layers have higher zoom factors, and therefore, represent the image at higher resolutions. The zoom factors can increase either linearly (e.g., *2x*, *3x*, *4x*, etc.) or exponentially (e.g., *2x*, *4x*, *8x*, etc.) Space for new data is allocated from a free list using a first-fit algorithm. Cache data blocks are replaced using a modified *Least Recently Used* (LRU) algorithm. Like standard LRU algorithms [Tanenbaum 1992], ImageEyes' replacement algorithm uses a queue structure to track the order in which cache blocks were last used. However, instead of using the first-fit block from the queue, the algorithm tries to preserve blocks containing data that overlaps with the requested region in the current operation. It does so by placing "in-use" locks on all such blocks. In finding space for a new data block, the replacement algorithm first releases and coalesces blocks without locks. After this process, if there is still insufficient free space for the new data, it splits locked blocks based on their potential value to the current operation. Blocks at the same layer as the current operation and with the smallest overlapping region are chosen first. When a locked block is split, the overlapping region is placed in a new block and marked as "in-use". The remaining portion is released and coalesced with other free blocks.

The client's cache can reduce the amount of data that is needed for zoom and pan requests in two ways. First, when a user moves to a new region and there are pieces of that region in the cache at the desired level or lower (higher zoom factor), then those pieces do not need to be transmitted. Their display can be computed directly from the cache. Second, when cache contains data at a higher layer for the requested region, the client only needs delta values, which are the difference between the aggregate color values of the higher layer and the exact color values for the requested layer. In many cases, these delta values fall into a small range, e.g. *-15* to *+16*, and therefore, can be packaged and compressed into fewer transmission packets.

The server's *image processor* is responsible for interpreting an incoming request and deciding the minimum data needed to satisfy that request. First, depending on the contents of the client's cache, it breaks the request up into non-overlapping sub-regions. Then, for each sub-region, it chooses how to package and compress the data. More specifically, it decides whether to send delta values or absolutes values. If the image processor decides to send delta values, it also determines how they should be striped, i.e. broken up in to smaller incremental values. Sending striped delta values allows for lossy image transmission and, in many cases, actually requires fewer transmission packets than sending absolute value. Sending striped delta values also results in a nice fade-in effect on the user's display, which gives the illusion of a quicker response time. Sending absolute values, however, is more appropriate when the delta values cover a large range and do not fit a normal curve.

The ImageEyes server is designed to support different storage formats for the high resolution images through format-specific *file I/O handlers*. At this point, we have only implemented a file I/O handler for Raw PPM, but plan to build handlers for JPEG, GIF, and TIFF in the near future.

3 Operational Overview

With the stand-alone ImageEyes client, the user starts a viewing session by executing the client program with parameters that specify a host machine and the name of an image. The client connects directly to the ImageEyes server running on the host machine and sends it an open request. The server returns some startup parameters directly to the client. With a web-based ImageEyes client, the user simply clicks on an ImageEyes link in their web browser. This causes an open request to be sent to the ImageEyes server via a *FastCGI interface* [Fig. 1] The ImageEyes server initiates a new session and returns the startup parameters to the web server, which in turn, sends them to the browser. The browser uses this information to activate the

client software. From that point on, the client software interacts directly with the ImageEyes server just like the stand-alone version.

Consider, as an example, a user who starts a viewing session for an image of a plant against a plain beige background. Although there are very few sharp edges in the image and the contrast is low, it contains sufficient detail to show hairs on the edges of the leaves. After opening the session, the server sends the client an initial view of the entire image at a *1x*-zoom factor. The client displays this data and stores it in the top layer of the cache. Next, the user zooms into the top-left corner of the image. This causes a request to be sent to the server for this region at a desired zoom factor, say *4x*. The request also tells the server that the client's cache only contains the top-layer data. The server interprets the request, reads the necessary data from the stored images, and decides on the best processing strategy. Because of the low contrast and lack of sharp edges, it would likely choose to use delta values, striped into increments ranging from *-7* to *+8*. Assuming that the network conditions are relatively good, it would also choose to use standard compression parameters and follow a transmission protocol that tolerates a small amount of loss before retransmission requests are issued.

Just after the client sends the request to the server, it starts computing the display based on the information in the cache. So, in this case, the user will immediately see a course-grain blow up of the top-left corner. The first 4x4 group of pixels will have the same initial value as the top layer's first pixel; the next 4x4 group will have the same initial value as the top layer's second pixel; and so on. The client can receive the response packets from the server at any time and in any order, as long as the request hasn't been superceded by a subsequent request. Since the packets contain delta values, the client simply adds them to the display and updates the cache. To the user, it appears as if the image is coming into focus.

Suppose the user now decides to pan a quarter of the display width to the right. The client creates and sends a new request to the server. The display is again updated based on the information in the cache. This time, the left portion of the image that is to be displayed is already in the cache at the correct layer. The right portion of the image is filled in with a course-grain blow up of the top layer in a fashion similar to the previous request. Because the request included the contents of the client's cache, the server knows that it only needs to send delta values for the right portion of the image. The server computes these delta values relative to the top layer and sends them to the client. The right portion of the image is filled in as the client receives the packets.

The user makes a final request to zoom in to the image. The display is updated from the contents of the cache so as much detail as possible is showing. Some portions of the image may contain data at the second layer while other portions may contain data at the top layer. Suppose that the client doesn't have enough free space in the cache to hold all of the data coming from the server. Some of the cache contents will be replaced based on the algorithm discussed earlier. The least important blocks are removed or split and the cache is updated with the new data.

4 Performance Evaluation

We will evaluate the performance of ImageEyes from several different perspectives. First, we will examine it on its own merits by testing both normal and worst-case scenarios for various kinds of images. The scenarios will consist of sequences of zoom and pan operations. The images will vary in size, number of colors, regularity (repetition of patterns within an image), and clarity. For each scenario and image, we will record the requests generated by the client and the sizes of the data blocks returned by the server. We will not measure transmission time, since it depends on other variables, like bandwidth and current network traffic.

Second, we will compare ImageEyes to several existing techniques, including non-interlaced images, interlaced images, predefined views, and fully partitioned images. Since the zoom and pan operations do not apply to all of these techniques in the same way, we will rate the relative effectiveness from a user's perspective. More specifically, we will look at the following issues:

- How much data is required before the user gets a glimpse of the image?
- How much data is required before the user has a complete overview of the image?
- How much storage space is required on the local system to view the image?
- What kinds of delays occur between zoom and pan requests?
- Was backtracking natural and effective?

5 Related Work

Much research has been done over the past few years on topics related to the ImageEyes project, including caching schemes, image compression, and image transmission. However, most of the work has focused on a specific area, like caching, and not on global optimization across multiple areas. Caching schemes have been extensively researched for operating systems and their properties are relatively well understood in this context [Tanenbaum 1992]. However, there are some significant differences between caching data dominated by sequential access patterns and caching images when there are three degrees of freedom in the movement through the data. Although the principle of *locality of reference* [Tanenbaum 1992] still applies, it is manifested in different ways. Nevertheless, common replacement algorithms, like LRU and NRU, and allocation algorithms, like first-fit and best-fit, can be adapted.

The areas of image compression and communication are also mature. However, current research efforts deal primarily with compression and transmission of entire images and therefore don't take into account cached data. Danskin, et al., describe an approach to image transmission that tolerates lost data with minimal distortion [Danskin et al. 1995]. Surveys of image compression techniques can be found in *Introduction to Data Compression* [Sayood 1996] and *Data Compression: Methods and Theory* [Storer 1988].

The most similar product to ImageEyes is one marketed by Hewlett Packard under the name *OpenPix Software Suite*® [OpenPix 1998]. This viewer is based on the FlashPIX™ file format [Lee 1998], which stores pre-defined views at different zoom levels. The OpenPix viewer is similar to the ImageEyes client in that the user can interact with a high-resolution image and avoid transmission of the entire image at one time. However, OpenPix seems to lack the kind of global optimization that is at the heart of ImageEyes' design. HP's application allows users to zoom in and out, but only at the layers stored in the FlashPix file. It also appears that OpenPix does not take advantage of client-side caching techniques. Zooming in and then back out results in retransmission of the original image, which leads us to believe that the data transferred from the server to the client are absolute values and not relative values based on a client's cache.

6 Summary

Because Image Eyes uses an integrated solution for caching, compression, communication, and display and because it works with existing Web technology, we believe that it will provide an effective means for viewing high-resolution images on the Web. For typical images and usage scenarios, ImageEyes should dramatically reduce the amount of information that needs to be transmitted, and thereby, provide near real-time image zooming and panning. Its worst case behavior should be at least as good as the full-partitioning approach described in the introduction.

To date, we have implemented version 1.0 of the server and client. This version includes the hierarchical caching and lossy data transmission schemes as well as an intelligent data compression technique. We are currently collecting test data to evaluate and optimize the performance of the ImageEyes system.

6 References

[Connolly 1998] Connolly, Dan (downloaded April 1998). HyperText Markup Language. World Wide Wide Consortium. http://www.w3.org/pub/WWW/MarkUp/

[Danskin et al. 1995] Danskin, J., G. Davis, and X. Song (1995). Fast Lossy Internet Image Transmission. *ACM Multimedia*, San Francisco, CA, November 1995. http://www.cs.dartmouth.edu/~jmd/decs/xsong/fliit/paper.html

[Sayood 1996] Sayood, Khalid (1996). *Introduction to Data Compression.* San Francisco: Morgan Kaufmann Publishers, 1996. ISBN 1-55860-346-8

[Storer 1988] Storer, J.A. (1988). *Data Compression: Methods and Theory*, Computer Science Press, Rockville, MD, ISBN: 0-88175-161-8.

[Tanenbaum 1992] Tanenbaum, A. S. (1992). *Modern Operating Systems*, Prentice Hall, 1992

[OpenPix 1998] OpenPix (downloaded April 1998). OpenPix Internet Imaging. http://www.image.hp.com/about.html

[Lee 1998] Lee, Ho John (downloaded April 1998). Too Many Pixels, Not Enough Bandwidth. http://www.image.hp.com/webguide.html

[HTTP 1998] HTTP Working Group (downloaded April 1998). Hypertext Transfer Protocol -- HTTP/1.1. Internet Engineering Task Force. http://www.ics.uci.edu/pub/ietf/http/

[NCSA 1998] NSCA (downloaded April 1998). Common Gatway Interface. The National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. http://hoohoo.ncsa.uiuc.edu/cgi/

[W3C 1998] W3C (downloaded April 1998). The World-Wide Web Consortium. http://www.w3.org/pub/WWW

U.S. Department of Education
Office of Educational Research and Improvement (OERI)
National Library of Education (NLE)
Educational Resources Information Center (ERIC)

NOTICE

REPRODUCTION BASIS

☑ This document is covered by a signed "Reproduction Release (Blanket) form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

☐ This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").